GR 98 P 2892

METHOD OF PROTECTING ENTRY ADDRESSES

Cross-Reference to Related Application:

This application is a continuation of copending International Application No. PCT/DE99/03169, filed October 1, 1999, which designated the United States.

Background of the Invention:

Field of the Invention:

The invention relates to a method of protecting entry addresses in computer programs.

For future smart cards it is intended that applications developed by third-party companies can call functions in the operating system. In this case, there could be a risk that these applications from third-party companies attempt to manipulate or sabotage other program parts. Such a possible attack could be performed by using a different entry address rather than using a valid function pointer to an operating system routine from the application. In consequence, the operating system code would not be carried out correctly and could, for example, provoke possible data losses resulting from a memory area being incorrectly overwritten.

One solution to overcome the above-described problem provides an entry point or vectoring to predefined addresses (so-called "gates") at fixed intervals. In this case, it is first of all necessary to check whether a gate address is located within the range of the permissible addresses for that module. If the result is positive, a jump is made to that gate, and a further jump is then made from there to the actual function. This is disadvantageous since a calculated jump must be carried out in this case, which leads to a so-called "peephole" in the CPU (Central Processing Unit) pipeline. Alternatively, the jump to the function can be carried out with a delay after a number of prioritized program instructions of that function. This leads to a difficult optimization problem for the compiler if a loop is initiated at the start of the function and it is necessary to continuously jump backward and forward between the code at the gate and the remote code.

Summary of the Invention:

It is accordingly an object of the invention to provide a method of protecting entry addresses which overcomes the above-mentioned disadvantages of the heretofore-known methods of this general type and which requires no calculated jumps and thus causes no peephole in the pipeline of the CPU instructions.

With the foregoing and other objects in view there is provided, in accordance with the invention, a method for protecting entry addresses, which includes the steps of:

identifying a permissible entry address by using a correlation of data, wherein the data are not provided within the same individual instruction; and

storing, in a memory cell, an address of a correlated data item directly before or directly after the permissible entry address.

The object of the invention is achieved by allowing a jump directly to permissible entry addresses, wherein the permissible entry addresses can be identified by a correlation of data which cannot occur within the same, individual instruction.

In this case, an organization of the program code allows the compiler or linker to ensure that only legal entry addresses satisfy this correlation. For example, the correlation process can be carried out by providing that the memory cell contains the address of correlated data items directly before or after the entry address.

A preferred mode of the invention provides that the memory cell contains a reference to the corresponding data entry in a protected list of legal entry addresses, the reference being provided directly before or after the entry address.

According to another mode of the invention, a direct jump to the permissible entry address is provided.

In this case, it is particularly preferable to carry out an automatic check when executing the function call, in order to determine whether the correlation of the data is satisfied.

With the objects of the invention in view there is also provided, a method for protecting entry addresses, which includes the steps of:

identifying a permissible entry address by using a correlation of data, wherein the data are not provided within the same individual instruction; and

providing the correlation of data as a correlation with program data in non-reserved memory areas.

It is particularly preferable to carry out an additional automatic check during execution of the function call in order

to determine whether the correlated data item is in the reserved memory area provided for this purpose.

If program instructions do not exceed a certain maximum number n of bytes, a further solution according to the invention can be used, wherein a specific no-operation code is provided which is used to avoid random correlations and which can be inserted retrospectively by the compiler or linker.

In this case, it is particularly preferable to carry out the correlation between code data items which are at least n bytes away from one another.

Furthermore, according to the invention, the entry address can be protected by the insertion of a specific byte sequence, which cannot occur within the regular code, for example by using a specific no-operation code.

Thus, according to the invention, the peephole in the pipeline can be avoided by jumping directly to the address of the function pointer. However, to do this, it is necessary to ensure that legal jump addresses are distinguished by non-local code correlations. The program code must be organized to allow the compiler or linker to ensure that only legal function jump addresses satisfy this correlation. "Non-local

correlation" in this case means a correlation of data items which cannot occur within the same, individual instruction.

Description of the Preferred Embodiments:

The following preferred embodiments of the invention are for example feasible:

1. Correlation with data in memory areas reserved for this purpose: A simple implementation could, for example, include the address of a correlated data item which, for example, once again corresponds to the legal entry address of the function, directly before the entry address in the memory cell. When the function call is executed, an automatic check can be carried out to determine whether this correlation is satisfied and/or whether the correlated data item is in the intended, reserved memory area. At first glance, the mechanism is very similar to the previous gate mechanism, but has the advantage that no calculated jump takes place, and the function instructions can be placed directly in the prefetcher of the pipeline. A peephole occurs only in the erroneous situation of an illegal vector.

2. Correlation with program data in non-reserved memory areas. This solution presupposes that the program instructions do not exceed a certain maximum number n of bytes. Data areas in the code segment longer than that must then be excluded.

Furthermore, this method is dependent on a specific no-operation code (SNOP) which is never used in the regular code and is inserted retrospectively by the compiler/linker solely to avoid random correlations. It is thus possible to distinguish between two different types of solution here:

a) The correlation is done between the code data items which are at least n bytes away from one another. The compiler or linker must in this case ensure that any random correlations in the code are avoided by introducing SNOP intermediate codes.

One possible implementation is as follows: directly before the entry address of the function, there is a value which is a function of the subsequent n+m ($m \geq 0$, otherwise arbitrary) bytes. Should this correlation be satisfied randomly anywhere in the code, then the compiler or linker must cancel this random correlation: since, according to the assumption, at least one real instruction will end in the sequence of n+m bytes, a series of SNOP instructions can be inserted after the end of such a real instruction, until the function value changes. Within certain limits, the function can in this case be chosen freely.

b) The entry address is protected by the insertion of a specific byte sequence which cannot occur within the regular code. One example of this is a sequence of OP-codes (SNOP).

Thus, according to the invention, the entry addresses for functions are protected by non-local code correlations which can occur only at the entry addresses.

This therefore advantageously avoids the gate mechanism, which results in a calculated jump and causes a peephole in the instruction pipeline.

Instead of this, a direct jump to the entry address is performed for jumping into in the function. The subsequent instruction can be loaded into the pipeline irrespective of whether the verification of the jump address turns out to be positive or negative. This therefore improves the efficiency of monitored function calls.